


PATENT

"EXPRESS MAIL" Mailing Label Number

EU516995723US

I HEREBY CERTIFY THAT THIS PAPER OR FEE IS BEING
DEPOSITED WITH THE U.S. POSTAL SERVICE "EXPRESS
MAIL POST OFFICE-TO-ADDRESSEE SERVICE" UNDER 37
CFR 1.10 ON THE DATE INDICATED BELOW AND IS
ADDRESSED TO: COMMISSIONER FOR PATENTS,
P.O. BOX 1450, ALEXANDRIA, VA 22313-1450 ON:

February 3, 2004
DATE OF DEPOSIT


SIGNATURE OF PERSON MAILING PAPER OR FEE

Jill Wolfe
NAME OF PERSON SIGNING

February 3, 2004
DATE OF SIGNATURE

A SYSTEM FOR EVOLUTIONARY ADAPTATION

Related Application

This application claims the benefit of US Provisional
Application No. 60/445,579, filed February 7, 2003.

5

Field of Invention

The present invention relates to a system for
evolutionary adaptation within a wireless network, and
10 more specifically, to a system for evolving services and
protocols within wireless network operation.

Background of the Invention

Dynamic Ad-Hoc Wireless Networks (DAHWNs) are a
subset of variable topology networks. The goal of
15 variable topology networks is to maintain message delivery
as the network topology varies. Network nodes should be
able to dynamically form transient networks. Nodes, which

may be located on rapidly moving platforms such as aircraft, should be able to join, leave, and re-join networks which may form at any time. Networks spontaneously form and their topologies may change rapidly or almost immediately. An additional challenge required by airborne and heterogeneous air and ground environments is the ability to provide predictable and optimized Quality of Service (QoS) of data transmission over variable topologies.

10 In order to provide predictable and optimal Quality of Service (QoS) in Dynamic Ad-Hoc Wireless Networks, network architecture must support dynamic adaptation to the rapidly changing environment. The degree to which the network must adapt is dependent on the rate of change of the topology. QoS requirements are most often stated in the form of an optimization problem with a cost function that is optimized by adaptation within the network. An applicable result from complexity theory, a No Free Lunch Theorem, expresses a limit on the ability of any single algorithm, or protocol, to meet QoS requirements. The No Free Lunch Theorem states that all algorithms perform exactly the same, searching for an extremum, when averaged over all cost functions. If a potentially good algorithm appears to outperform poor algorithm on some cost

functions, then there exist exactly as many functions where the apparent poor algorithm will outperform the good algorithm. In other words, no single algorithm, or ad-hoc protocol, can optimize all potential QoS requirements.

5 There are two forms of adaptation of protocols: 1) an algorithm that remains fixed, but includes tunable parameters and 2) an algorithm whose fundamental operation changes. Most conventional theory has focused upon the fixed, but tunable adaptation. In other words, current
10 research is seeking a fixed algorithm with enough degrees of freedom such that optimal operation may be found by tuning a fixed set of parameters. This may be due in part to the difficulty in breaking away from the fixed operation of the Internet Protocol that has a strong grip
15 on the mind-set of most researchers. Great potential exists in examining the latter form of adaptation, particularly in light of the implication of the No Free Lunch Theorem which indicates that simply tuning a given algorithm will not be as optimal as changing the algorithm
20 itself.

Two high-level frameworks that are flexible and customizable enough to allow dynamic change in algorithmic content within networks are: Programmable Networks and Active Networks. A Programmable Network allows control

software of the network to be dynamically reprogrammed. An Active Network is an extreme form of programmable network that allows code and data to travel through the network, often in the same packet structure. Active
5 packet code may execute on any node along the path that the packet travels. Active networks may service both mobile and ad-hoc networks. One challenge that must be addressed is the mismatch among adaptation of individual layers of Internet Protocol (IP) and improving the
10 adaptation to suit the characteristics of wireless and ad-hoc network environments.

The most significant gap that has been identified with regard to adaptation within ad-hoc networks is the lack of synergistic adaptation among network layers.
15 Early network implementation focused upon network layering as a mechanism for partitioning computer communications into a set of tractable sub-tasks.

Layering has resulted in many forms of adaptation occurring simultaneously within the network. At times,
20 adaptation in one layer (e.g., congestion control) may occur in a manner antithetical to adaptation in another layer (e.g., route repair). A "meta"-adaptation view, namely how adaptive mechanisms work together, is extremely

important for an ad-hoc network environment, but is currently lacking.

One conventional attempt to correct this deficiency is Explicit Link Failure Notification. Congestion and
5 routing each try to adapt based upon limited knowledge of each other, resulting in sub-optimal global behavior. Another example of sub-optimal adaptation behavior is MAC to IP layer address resolution.

Non-layered ad hoc communication in sensor networks
10 may also provide useful information. A sensor network tends to assume large numbers of constrained sensor devices that transmit asymmetrically to a central location. However, ad hoc routing must be implemented on the sensors using as little power and processing as
15 possible. This has resulted in fewer network layers and better in-network utilization via active networking.

It would be desirable for wireless ad-hoc networks to minimize network misconfiguration, bandwidth and processor misallocation, faults caused by distributed denial of
20 service, virus attacks, sub-optimal traffic shaping, sub-optimal routing, sub-optimally fused data, sub-optimal link quality, miscomposed modeling, and sub-optimally tuned components. Further, conventional ad-hoc wireless networks do not have the capability of "self-healing". If

a conventional network encounters a situation that exceeds its predefined tolerances, the conventional network will likely exhibit catastrophic failure.

Summary of the Invention

5 A system in accordance with the present invention operates a wireless ad hoc network. The system includes a plurality of nodes and an active packet. The active packet implements a genetically programmed adaptation of one of the plurality of nodes in response to a change of
10 condition of the one node of the plurality of nodes.

 A computer program product in accordance with the present invention evolutionarily adapts a network. The computer program product includes a first instruction for implementing a genetically programmed adaptation of one of
15 a plurality of nodes in response to a change of condition of the one node of the plurality of nodes and a second instruction for injecting a functional unit into the active packet. The first instruction is executed by an active packet.

20 A method in accordance with the present invention adapts a network. The method includes the steps of: operating a plurality of nodes with active packets; implementing a genetically programmed adaptation of one of the plurality of nodes in response to a change of

condition of the one node of the plurality of nodes;
executing the operating step by the active packet;
injecting a functional unit into the active packet; and
probabilistically selecting two parental programs based on
5 fitness.

Brief Description of the Drawings

The foregoing and other features of the present
invention will become apparent to one skilled in the art
to which the present invention relates upon consideration
10 of the following description of the invention with
reference to the accompanying drawings, wherein:

Fig. 1 is a schematic representation of an example
system for use with the present invention;

Fig. 2 is a schematic representation of part of a
15 network for use with a system in accordance with the
present invention;

Fig. 3 is a schematic representation of another
example system in accordance with the present invention;

Fig. 4 is a schematic representation of part of still
20 another example system in accordance with the present
invention;

Fig. 5 is a schematic representation of yet another
example system in accordance with the present invention;
and

Fig. 6 is a schematic representation of still another example system in accordance with the present invention.

Description of an Example Embodiment

A system 10 in accordance with the present invention
5 (Fig. 3) includes an active network packet for
implementing genetically programmed adaptation to respond
to variable and unforeseen network conditions. The active
packet may be represented by a nucleus 101 with each
network node 100 being represented by a cell containing
10 the nucleus (Fig. 3). The nucleus 101 may contain a
population 111 of chromosomes (i.e., strings of functional
units, etc.). Functional Units may be predefined code
units within the active packets. A genetic network
programming operation may begin with the injection of the
15 functional units, or basic building blocks of genetic
material, into the network. This genetic material may be
injected into each active node of the network. The
genetic material remains inactive until a particular
fitness function is injected into the network. Receipt of
20 this particular fitness function causes evolution.
Evolution continues as long as the network continues to
operate.

The system 10 defines the interoperability
requirements for an active ad-hoc network by

evolutionarily adapting the service of the network.

Networks are typically divided into fixed layers to allow for specialization and ease of development. However, different optimization techniques and strategies may be

5 used within different layers, sometimes leading to conflicting goals and lack of optimal convergence. The goal of the system 10 is to define an optimization service that is integrated with the network. Thus, common optimization and fitness goals may be enabled across all
10 layers. The system 10 also allows the automated downward growth of protocol stacks such that as the reliance upon a fixed infrastructure is minimized.

The system 10 injects algorithmic information (i.e., executable code, etc) into the network. Active or
15 programmable capability optimizes this injection. While active ad-hoc network generic adaptation may be standardized via a vehicle other than active packets, active networking allows for convenient implementation of algorithmic change within the network.

20 The system 10 provides an optimization service that is integrated throughout the network architecture. In order to achieve the goal of an infrastructure-less ad-hoc network, the system 10 is highly adaptable. The

adaptation framework of the system 10 minimizes reliance upon a fixed infrastructure.

The system 10 utilizes a DAHWN layering (Fig. 2) that is not a conventional network stack. In particular, all
5 of the components above the Active Network Execution Environment are Active Applications and do not require the services of other Active Applications lower in the stack. While the Evolutionary Adaptation Service (EAS) provided by the system 10 may be located at the top of the stack,
10 the EAS is available to support all of the components in the stack. In addition, the EAS does not rely on all the stack elements beneath it. For example, EAS does not require routing (which may appear below it in the stack), unless it is being remotely injected. Also, EAS is
15 available to support services that appear below the EAS in the stack. Specifically, a Policies block may require the EAS to provide optimal solutions to security problems or to evolve components that meet specified security requirements (i.e., components with given levels of
20 complexity, etc.).

Another example may be complexity probes located deep within the Active Execution Environment (lower in the stack) may use an evolutionary complexity estimator that is provided by the EAS. The system 10 exists within the

DAHWN architecture (Fig. 2). The active ad-hoc network genetic adaptation service of the system 10 runs as an active application on an active node. The EAS is independent of the underlying architecture of the DAHWN.

5 The Node Operating System (NOS) may run one or more Execution Environments (EE) within a protocol stack of a DAHWN. Multiple active applications may execute in any Execution Environment. The protocol stack provides the architecture of an active network overlay of protocols.
10 This overlay scheme uses the Active Network Encapsulation Protocol (ANEP) as a conduit for the underlying network. The genetic adaptation system 10 in accordance with the present invention executes alongside other active applications and interacts with any packet managed entity
15 to provide optimization and adaptation services.

EAS is an active ad-hoc Evolutionary Adaptation Service. EAS may provide mathematical optimization results and genetically modify itself to meet a specific fitness criteria. Small-State is an information cache
20 that may generally be created at network nodes 11 (Fig. 1), intended for use by executable components of the same application. Global-State is an information cache created at network nodes 11, intended to be used by executable components of different applications. Active Application

(AA) is an active network protocol or service that is injected into the network in the form of active packets. The active packets are executed within the EE. Active Network allows executable code to be injected into the
5 nodes of the network and allows the code to be executed at the nodes. Active Packet is the executable code that is injected into the nodes of an active network. Node Operating System is the active network operating system. The supporting infrastructure on intermediate network
10 nodes 11 supports one or more execution environments.

A Mutation Operation creates a single parental program probabilistically selected from the population based on fitness. A mutation point is randomly chosen, the subtree rooted at that point is depleted, and a new
15 subtree is grown there using the same random growth process that was used to generate the initial population. This type of asexual mutation operation is typically performed sparingly (having a low probability during each generation of the run).

20 Cross-Over is a sexual recombination operation with two parental programs probabilistically selected from the population based on fitness. The two parental programs are usually of different sizes and shapes. A crossover point is randomly chosen in the first parental program and

a crossover point is randomly chosen in the second parental program. A subtree is rooted at the crossover point of the first, or receiving, parental program and is deleted and replaced by the subtree from the second, or contributing, parental program. Crossover is the predominant operation in genetic programming (and genetic algorithms) and is performed with a high probability..

A Reproduction Operation copies a single individual, probabilistically selected operation based on fitness, into the next generation of the population. A Functional Unit (FU) is an active packet containing code that is capable of operating upon other active packets. The Functional Unit has a well-defined input port and output port through which active packets may flow as the active packets are being executed. Functional Units are the building blocks of a Chromosome. A Functional Unit may range in complexity from very simple math operations to very complex packet modifications upon active packets flowing through the Functional Unit.

A chain of Functional Units has an input port and output port. Chains of Function I/O ports form a Chromosome. Evolution occurs by changing the ordering of the Functional Units.

A Nucleus is the evolutionary control code and the

initial set of Functional Units. Evolutionary Control Code includes evolutionary and genetic programming mechanisms. The Evolutionary Control Code is the particular implementation injected into active nodes.

5 A Fitness Function may define the evolutionary goal. The system 10 continuously evaluates genetic material (chromosomes) to determine how well the genetic material meets the specified fitness criteria. The Fitness Function may be user-defined. However, the Fitness
10 Function must return a relatively high value for 'fit' genetic material and a relatively low value for 'less fit' genetic material.

 The above-discussed parameters may be the minimum necessary elements for active ad-hoc network genetic
15 adaptation. Fitness Functions may thereby be injected into the network allowing optimized static and algorithmic results. Such Fitness Functions may be executed seamlessly with other Fitness Functions within the network.

20 The system 10 conforms with existing standards when and where possible. The system 10 facilitates a gradual transition to an active and programmable networking paradigm. The system 10 performs Common In-line Optimization for introducing the use of a

common, cross-layer optimization technique. An evolutionary algorithm is an umbrella term used to describe a computer-based problem solving system using computational models.

5 The system 10 may use a variety of EVOLUTIONARY ALGORITHMS. Some examples are GENETIC ALGORITHMS, EVOLUTIONARY PROGRAMMING, EVOLUTION STRATEGIES CLASSIFIER SYSTEMS, and GENETIC PROGRAMMING. These all share a common conceptual base of simulating the evolution of
10 individual structures via processes of SELECTION, MUTATION, and REPRODUCTION. The processes may depend on the perceived performance of the individual structures as defined by an environment.

More specifically, EASS maintain a population of
15 structures that evolve according to rules of selection and "search operators", or genetic operators such as recombination and mutation (Fig. 4). Each individual in the population may receive a measure of fitness within the environment. Reproduction receives attention on high
20 fitness individuals, thus exploiting the available fitness information. Recombination and mutation perturb those individuals, providing general heuristics for Exploration. Although simplistic from a biologist's viewpoint, these

algorithms are sufficiently complex to provide robust and powerful adaptive search mechanisms.

This genetic programming may start with a "primordial ooze" of randomly-generated computer programs. The set of
5 functions that may appear at the internal points of a program tree may include ordinary arithmetic functions and/or conditional operators. The set of terminals appearing at the external points typically include the program's external inputs (such as the independent
10 variables X and Y) and random constants (such as 3.2 and 0.4). The randomly created programs typically have different sizes and shapes.

A main generational loop (Fig. 4) of a run of genetic programming may consist of a fitness evaluation (i.e.,
15 Darwinian selection) and genetic operations. Each individual program in the population may be evaluated to determine how fit that individual program is at solving the problem at hand. Programs may then be probabilistically selected from the population based on
20 fitness to participate in the various genetic operations, with re-selection allowed.

While a more fit program has a better chance of being selected, even individuals known to be unfit are allocated some trials in a mathematically principled way. Thus,

genetic programming is not a purely greedy hill-climbing algorithm. The individuals in the initial random population and the offspring produced by each genetic operation are all syntactically valid executable programs.

- 5 After many genetic operations, a program may emerge that solves, or approximately solves, the problem at hand.

The system 10 injects the Fitness Function onto all reachable nodes. The Fitness Function, a user-defined function, may read either Simple Network Management Protocol (SNMP) object values or EE Postevents. The result of Fitness Function execution may be a single fitness metric that is used by the Nucleus for evolutionary control. Postevents make only those metrics accessible that are local to the node. A SNMP interface 15 may allow network wide metric access.

The Fitness Function, once injected into a node, places itself in Small-State. The Fitness Function is picked up by the Nucleus and used to guide the evolution of the system 10. Chromosomes may accept active packets 20 flowing through the node and act upon those packets. This is determined by the definition of the Functional Units. Example actions include delaying packets, changing packet forwarding, and measuring packet characteristics.

Chromosome Strands may be composed of chains of Functional Units. An individual Functional Unit may be derived from an FU class and over-ride the function of the Functional Unit. The FU class ensures that the Functional Unit active packet has the properties required to chain I/O together in a single Chromosome Strand. The Functional Unit has well-defined I/O ports through which other active packets may flow and may execute user-defined actions in the Functional Unit.

10 The goal of the system 10 is to develop in-network self-composition of protocols and services. The system 10 follows a close analogy with biological evolutionary techniques such as Genetic Algorithms. Functional Units, or building blocks of code, may be injected into the network. In addition, a Fitness Function, defined by the user, may be injected into the network. The Functional Units evolve to maximize the Fitness Function.

20 The system 10 may include an EAS prediction framework for enforcing certain minimal requirements on the execution environment. The EE must provide an information cache, or Small State, to enable information exchange between active packets. The EE may also provide an information cache, or Global State, to enable an EAS prediction framework to communicate with a predictively

managed active application for querying the current state of the active application. The EE must be able to store and query both Small State and Global State, if Global State is implemented. The EE should provide appropriate
5 access control mechanisms to both Small State and Global State, if Global State is implemented.

The EE must provide an interface that enables both the active ad-hoc network genetic adaptation values and the values of the actual component being managed to
10 publish their state to an SNMP. This enables the EAS prediction framework to store the predicted state in a well-known format and also enables legacy SNMP tools to query the predicted state using SNMP operations. Additionally, the system 10 may also update its current
15 state using SNMP, which a Logical Process will be able to query.

In a particular implementation of such an interface, a generic SNMP agent coded as an active application may be injected into the active nodes. The agent creates a
20 'Global State' on an active node with a well-known name. The agent reads information coded in a well known format that has been written to the 'Global State' and publishes it to the SNMP. Any active application that wishes to advertise its state uses an interface that enables it to

store its information in the well-known 'Global State' in the given format.

The SNMP agent and the active application may use special interfaces to implement messaging between them. A
5 Message Packet may be the basic unit of inter-application communication. Each message consists of a message type.

The active application should send a message of the valid message type to the SNMP agent to perform the required operation. On receipt of a message, the SNMP
10 agent should attempt to perform the requested operation. The SNMP agent then responds with an acknowledgement message in a particular format.

The status code may have one of the following values:
OK: indicate successful operation; ERR_DUPENTRY: if for
15 a MSG_ADD operation, an object identifier of given name already exists; and ERR_NOSUCHID: if for a MSG_UPDATE operation, an object identifier of given name does not exist. The status message may be any descriptive string explaining the nature of the failure or should be
20 "Success" for a successful operation.

Models injected into the system 10 may allow network state to be predicted and efficiently propagated throughout the active network enabling the system 10 to operate simultaneously in real time as well as project the

future state of the system. Network state information, such as load, capacity, security, mobility, faults, and other state information with supporting models, is automatically available for use by the system 10 with
5 current values and predictive values. In one example, sample load and processor usage prediction applications have been validated using an Atropos Toolkit. The toolkit's distributed simulation infrastructure takes advantage of parallel processing within the network since
10 computation occurs concurrently at all participating active nodes. The example network may be queried in real time to verify the prediction accuracy. Measures, such as rollbacks, are taken to keep the simulation in line with actual performance.

15 In accordance with the present invention, a computer program product 500 evolutionarily adapts a network (Fig. 5). The computer program product 500 includes a first instruction 501 for implementing a genetically programmed adaptation of one of a plurality of nodes in response to a
20 change of condition of the one node of the plurality of nodes and a second instruction 502 for injecting a functional unit into the active packet. The first instruction 501 is executed by an active packet.

In accordance with the present invention, a method 600 adapts a network (Fig. 6). The method 600 includes the steps of: operating 601 a plurality of nodes with active packets; implementing 602 a genetically programmed adaptation of one of the plurality of nodes in response to a change of condition of the one node of the plurality of nodes; executing 603 the operating step by the active packet; injecting 604 a functional unit into the active packet; and probabilistically selecting 605 two parental programs based on fitness.

In order to provide a context for the various aspects of the present invention, the following discussion is intended to provide a brief, general description of a suitable computing environment in which the various aspects of the present invention may be implemented. While the invention has been described above in the general context of computer-executable instructions of a computer program that runs on a computer, those skilled in the art will recognize that the invention also may be implemented in combination with other program modules.

Generally, program modules include routines, programs, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate

that the inventive methods may be practiced with other computer system configurations, including single-processor or multiprocessor computer systems, minicomputers, mainframe computers, as well as personal computers, hand-
5 held computing devices, microprocessor-based or programmable consumer electronics, and the like. The illustrated aspects of the invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked
10 through a communications argument model. However, some, if not all aspects of the invention can be practiced on stand-alone computers. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

15 An exemplary system for implementing the various aspects of the invention includes a conventional server computer, including a processing unit, a system memory, and a system bus that couples various system components including the system memory to the processing unit. The
20 processing unit may be any of various commercially available processors. Dual microprocessors and other multi-processor architectures also can be used as the processing unit. The system bus may be any of several types of bus structure including a memory bus or memory

controller, a peripheral bus, and a local bus using any of a variety of conventional bus architectures. The system memory includes read only memory (ROM) and random access memory (RAM). A basic input/output system (BIOS),
5 containing the basic routines that help to transfer information between elements within the server computer, such as during start-up, is stored in ROM.

The server computer further includes a hard disk drive, a magnetic disk drive, e.g., to read from or write
10 to a removable disk, and an optical disk drive, e.g., for reading a CD-ROM disk or to read from or write to other optical media. The hard disk drive, magnetic disk drive, and optical disk drive are connected to the system bus by a hard disk drive interface, a magnetic disk drive
15 interface, and an optical drive interface, respectively. The drives and their associated computer-readable media provide nonvolatile storage of data, data structures, computer-executable instructions, etc., for the server computer. Although the description of computer-readable
20 media above refers to a hard disk, a removable magnetic disk and a CD, it should be appreciated by those skilled in the art that other types of media which are readable by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, and the

like, may also be used in the exemplary operating environment, and further that any such media may contain computer-executable instructions for performing the methods of the present invention.

5 A number of program modules may be stored in the drives and RAM, including an operating system, one or more application programs, other program modules, and program data. A user may enter commands and information into the server computer through a keyboard and a pointing device,
10 such as a mouse. Other input devices (not shown) may include a microphone, a joystick, a game pad, a satellite dish, a scanner, or the like. These and other input devices are often connected to the processing unit through a serial port interface that is coupled to the system bus,
15 but may be connected by other interfaces, such as a parallel port, a game port or a universal serial bus (USB). A monitor or other type of display device is also connected to the system bus via an interface, such as a video adapter. In addition to the monitor, computers
20 typically include other peripheral output devices (not shown), such as speaker and printers.

The server computer may operate in a networked environment using logical connections to one or more remote computers, such as a remote client computer. The

remote computer may be a workstation, a server computer, a router, a peer device or other common network node, and typically includes many or all of the elements described relative to the server computer. The logical connections
5 include a local area network (LAN) and a wide area network (WAN). Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the internet.

When used in a LAN networking environment, the server
10 computer is connected to the local network through a network interface or adapter. When used in a WAN networking environment, the server computer typically includes a modem, or is connected to a communications server on the LAN, or has other means for establishing
15 communications over the wide area network, such as the internet. The modem, which may be internal or external, is connected to the system bus via the serial port interface. In a networked environment, program modules depicted relative to the server computer, or portions
20 thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

In accordance with the practices of persons skilled in the art of computer programming, the present invention has been described with reference to acts and symbolic representations of operations that are performed by a
5 computer, such as the server computer, unless otherwise indicated. Such acts and operations are sometimes referred to as being computer-executed. It will be appreciated that the acts and symbolically represented operations include the manipulation by the processing unit
10 of electrical signals representing data bits which causes a resulting transformation or reduction of the electrical signal representation, and the maintenance of data bits at memory locations in the memory system (including the system memory, hard drive, floppy disks, and CD-ROM) to
15 thereby reconfigure or otherwise alter the computer system's operation, as well as other processing of signals. The memory locations where such data bits are maintained are physical locations that have particular electrical, magnetic, or optical properties corresponding
20 to the data bits.

It will be understood that the above description of the present invention is susceptible to various modifications, changes and adaptations, and the same are intended to be comprehended within the meaning and range

of equivalents of the appended claims. The presently disclosed embodiments are considered in all respects to be illustrative, and not restrictive. The scope of the invention is indicated by the appended claims, rather than
5 the foregoing description, and all changes that come within the meaning and range of equivalence thereof are intended to be embraced therein.